INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

UMI

A Bell & Howell Information Company
300 North Zeeb Road, Ann Arbor MI 48106-1346 USA
313/761-4700 800/521-0600

**SOFTWARE DEVELOPMENT PROCESS AND STRATEGIES USED TO
EXPEDITE SOFTWARE DEVELOPMENT**

by

Suparna Kurdi, B. Eng. (Computer Science)

**A thesis submitted to
the Faculty of Graduate Studies and Research
in partial fulfilment of
the requirements for the degree of
Master of Management Studies**

**School of Business
Carleton University
OTTAWA, ONK1S5B6**

**May 13, 1997
© Copyright
1997, Suparna Kurdi**

National Library
of Canada

Acquisitions and
Bibliographic Services

395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque nationale
du Canada

Acquisitions et
services bibliographiques

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence

Our file Notre référence

The author has granted a non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

L'auteur conserve la propriété du droit d'auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

0-612-22136-9

The undersigned recommend to the Faculty of Graduate Studies
and Research acceptance of the thesis

**SOFTWARE DEVELOPMENT PROCESS AND STRATEGIES USED TO
EXPEDITE SOFTWARE DEVELOPMENT**

Submitted by Suparna Kurdi, B. Eng.
In partial fulfillment of the requirements for the degree of
Master of Management Studies

A handwritten signature in cursive script, appearing to read 'Ukumar', is written above a horizontal line.

Thesis Supervisor

A handwritten signature in cursive script, appearing to read 'Indu Kumar', is written above a horizontal line.

Director, School of Business

*Carleton University
May, 1997*

ABSTRACT

This study investigates the software development process in Canadian and Indian high technology firms. Furthermore, it determines the strategies that are considered to be effective in reducing the software development time.

From the interview results, two models were devised to describe the software development process. One model describes the software development process followed by Canadian high technology firms and the other describes the process followed by Indian high technology firms. The results of the self-administered questionnaire reveal that application of prototyping approach, application of object-oriented approach, use of project management techniques, and verification of requirements were perceived to be the most effective strategies in reducing the software development time.

ACKNOWLEDGMENTS

I would like to express my gratitude to my supervisor, Dr. Uma Kumar, for providing continuous insightful guidance during the research and writing stages of the thesis. I also appreciate the concrete input provided to my thesis by my committee members, Dr. Vinod Kumar and Dr. Darren Meister.

My special thanks go to Jean Blair, who has always provided me with the best possible administrative assistance.

Finally, I would like to dedicate this thesis to my parents, who are my constant source of inspiration.

TABLE OF CONTENTS

	ABSTRACT	ii
	ACKNOWLEDGMENTS	iii
	TABLE OF CONTENTS	iv
	LIST OF TABLES	vi
	LIST OF FIGURES	vii
1	INTRODUCTION	1
2	LITERATURE REVIEW	7
2.1	SOFTWARE DEVELOPMENT PROCESS	7
2.1.1	SOFTWARE DEVELOPMENT MODELS	9
2.1.1.1	THE WATERFALL MODEL	14
2.1.1.2	THE SPIRAL MODEL	18
2.2	STRATEGIES TO REDUCE SOFTWARE DEVELOPMENT CYCLE TIME	23
2.3	<i>A PRIORI</i> MODEL	31
3	RESEARCH OBJECTIVES	34
4	METHODOLOGY	35
4.1	RESEARCH DESIGN AND STRATEGY	35
4.2	SAMPLE SIZE AND SELECTION	36
4.3	FIELD WORK	37
4.4	PRELIMINARY INVESTIGATION	37
4.5	DATA COLLECTION AND ANALYSIS	38

TABLE OF CONTENTS

	DISCUSSION OF RESULTS	
5	SOFTWARE DEVELOPMENT MODEL	42
5.1	SOFTWARE DEVELOPMENT PROCESS MODEL	42
5.1.1	DETAILED DESCRIPTION OF THE STAGES IN THE MODEL	43
5.1.2	FORWARD AND FEEDBACK LINKAGES AMONG THE STAGES	50
5.1.3	SOFTWARE DEVELOPMENT MODELS IN CANADA AND INDIA	57
6	STRATEGIES USED TO REDUCE SOFTWARE DEVELOPMENT TIME	61
7	CONCLUSIONS	79
	LITERATURE CITED	85
	APPENDIX	
	APPENDIX I: REQUEST LETTER GIVEN TO RESPONDENTS PRIOR TO THE INTERVIEW	90
	APPENDIX II: QUESTIONNAIRE FOR THE TWO RESEARCH OBJECTIVES	92

LIST OF TABLES

TABLE	DESCRIPTION	
1	TYPES OF SOFTWARE PROJECTS STUDIED	44
2	THE LINKAGES EXISTING BETWEEN THE STAGES OF THE MODEL	53
3	PRESENCE OF LINKAGES AMONG VARIOUS STAGES OF MODEL	54
4	MEAN EFFECTIVENESS OF STRATEGIES	65
5	USE OF VARIOUS STRATEGIES	67

LIST OF FIGURES

FIGURE	DESCRIPTION	
1	THE WATERFALL MODEL	15
2	THE SPIRAL MODEL	20
3	<i>A PRIORI</i> MODEL	32
4	SOFTWARE DEVELOPMENT MODEL FOLLOWED IN CANADA	55
5	SOFTWARE DEVELOPMENT MODEL FOLLOWED IN INDIA	56
6	MEAN EFFECTIVENESS OF STRATEGIES	68

1. INTRODUCTION

In today's information age, software development plays a very important role in the smooth functioning of high technology industries. In such industries, software interprets and manipulates information in order to obtain desired functions and performance. In terms of computer terminology, software is a program or a set of programs that resides in a computer memory in the form of binary information. Software acts as an interface between humans and machine or rather between programs, in order to facilitate easy retrieval, manipulation, storage, and transfer of data or programs. Software also incorporates data structures in order to manipulate desired information.

Software development has attracted the imagination of scientists and researchers throughout the world. The United States of America and Canada, leading software developers, spend billions of dollars each year on software. In Canada, large multinational and national companies develop software purchased by industries throughout the world.

Software production is an intellectual process requiring engineering, imagination and intellect. In a competitive atmosphere, cost, quality and time are major considerations. To reduce the cost of production and to remain competitive, the western world is tapping Asian intelligentsia, available at lower costs. India is one of these sources. In the process, India is developing as one of the largest manufacturers of software for the world market. The cost of labor is less than a fourth of cost involved in the United States and Canada (Samyukta Karnataka, 1996).

The purpose of this research was to understand the software development process and to investigate the strategies being adopted by high technology industries to reduce time taken for the development of software to enable themselves to remain ahead in today's global

competitive market. Canadian and Indian high technology firms were considered for the study.

There are a number of Canadian and American companies using monetary advantages and collaborating with Indian counterparts. For example, a major Ottawa based business unit of Nortel that manufactures software for international markets is collaborating with Infosys located at Koramangala in Bangalore, India.

Ottawa-based Canadian companies like Corel, Newbridge and Cognos involved in software manufacturing for computers are world leaders. The Ottawa Silicon Valley is considered to be a major force, involved in the best and sought after software used all over in Canada and worldwide. They are also using worldwide intelligentsia available at low cost for developing competitively priced software.

Bangalore, the remote capital of Karnataka in Southern India formerly known as an overgrown village with a temperate climate is buzzing with commercial activity and is becoming the world's second largest software capital. Citibank, American Express, General Electric, IBM, Reebok, Texas Instruments, Hewlett Packard, Compaq Computers are some of the U.S. based world renowned companies who depend on software engineers in Bangalore and other centers in India for software development. The major attractions for American companies for using Bangalore as the personnel development point are low wages, high intellect, and worldwide comparable services of engineers (Samyukta Karnataka, 1996).

Ever increasing competition and super-fast technological changes emphasize the importance of understanding and following a conceptual order of software development. The conceptual order is the basis for development of efficient software.

The development of software entails the following stages: system and software requirements, preliminary design, detailed design, code and debug, test and pre-operations, operations and maintenance. The characteristics of a truly effective software development model are that it is predictable and the cost estimates and schedule commitments can be met with reasonable consistency. The quality of the resulting products generally meet user needs if the software development model is effective.

Despite continuous efforts to improve management and control, problems continue to exist in software development projects (Phan, Vogel and Nunamaker, 1995). In spite of the efforts to improve the processes and techniques used in the development of software, projects continue to suffer problems in meeting user expectations, schedule and budget. A software project is a highly people-intensive effort that spans a very lengthy period, with fundamental implications on the work and performance of many different classes of people (Boehm and Ross, 1989). In software development, users and producers usually both play an important role in the development process.

Over the years, researchers have studied the hardware development process in depth (Shina, 1991, Urban and Hauser, 1980). However, there is a lack of studies of the software development process. The processes followed in the hardware and software industry cannot be equated due to some major differences in their development processes (Mathur, 1993). One of the prominent differences is that when hardware is built, the creative ideas are converted into a physical form. Software is different as it captures information, produces information and is itself information. Software is a logical rather than a physical system element. Moreover, it can be said that in software, the quality of the program code decides the overall quality of the software. Also, much of the effectiveness of software depends on its computational complexity in terms of time, and space of RAM, and disks. Certain programming structures may not work well in certain time frames. A

software product may sometimes work perfectly with 100 records but may not work when used with 10000 records. Many system developers expect software to compensate for the deficiencies in the hardware design of a system. Software does not physically wear out as a hardware product's components do. It does not degrade from use. However, software has to be maintained throughout its life-span. These differences lead to the conclusion that the development processes of hardware and software products vary to a great extent.

The primary functions of a software development process model are to portray the order of the stages involved in software development and evolution and to establish the transition criteria for the next stage (Boehm, 1995). These include completion criteria for the current stage and entrance criteria for the next stage. The software industry is ever developing and in order to stay ahead in the competition, firms have to follow software development process models that will lead to the best product with the least expenditure in terms of time and money. Software development process models are important because they provide guidance on the order in which the project's major tasks should be carried out. Many software projects, have come to grief because they pursued their various development and evolution phases in the wrong order (Boehm, 1995). Software development methodologies assist in transforming the development of the software into a structured, engineered and managed approach that involves the user throughout the process. While there is no shortcut to getting it right and getting it right the first time, with good management skills being applied to the development of software, the outcome of the finished product can be predicted in terms of technical content, time-scale and cost (Pilkington, 1996). Thus, a deeper understanding of the software development process and ways to improve its efficiency are critical issues faced by firms.

Therefore, the *main purpose* of this research is to understand the practices followed in the software development process.

Software development starts off with the identification of an objective, usually a problem or an opportunity and is terminated with a product that solves or fulfills it. Boehm (1995) defines software engineering as "the practical application of scientific knowledge in the design and construction of computer programs (software) and the associated documentation required to develop, operate, and maintain them". There are two ways of fulfilling a software requirement. It can be fulfilled by using some other existing software package that will run the required application or by developing new software from scratch. This research considered only the latter.

An important issue in software is the development time, which is the time taken for the development of software from requirements specification to actual implementation of the product. Users will not look too kindly to delays in the development of software and these delays can lead to financial and reputational damage to the developer. The success of a software also depends on the time it takes to enter the market. There is also a strong possibility that as time elapses the user's requirements might change leading to more delays.

Certain strategies that help reduce the software development time are cited in the literature. The most commonly cited strategies to reduce the software development time in the literature are the application of prototyping approach, application of object oriented approach, use of reusable code and application packages, application of incremental development approach, use of CASE/automated tools, use of advanced project management techniques, use of structured methods and techniques, and verification of requirements. All these strategies are discussed in the literature review section.

Application of prototyping approach, object oriented approach, and incremental development approach, use of CASE/automated tools, and the use of structured methods and techniques are strategies that lay emphasis on using technological know-how in order to reduce the software development time. Use of reusable code and application packages are strategies that advocate the use of existing code in building new software which will help save time. Use of advanced project management techniques and verification of requirements advocate the use of good project management skills to reduce the software development time.

The *ancillary purpose* of this study, therefore, is to investigate the strategies being adopted by high technology firms to reduce the software development time in order to stay ahead in today's global competitive market.

This research considered software development projects undertaken in high technology firms. Today, high technology firms are involved in various projects concerning software development. High technology firms were selected since they gave a wider outlook to the research than just software development firms would have.

Section 2 of this thesis discusses the literature review conducted. Section 3 explains the research objectives and section 4 describes the methodology adopted for the study. Sections 5 and 6 discuss the results. Finally, section 7 discusses the conclusions arrived at.

2. LITERATURE REVIEW

The literature review for this research is divided into three sections. Section 2.1 describes the software development process models that are cited in the literature and also the limitations of the existing models. Section 2.2 covers the literature on strategies to help reduce the software development time. Finally, Section 2.3 focuses on an *a priori* software development model developed as a basis for this research.

2.1 SOFTWARE DEVELOPMENT PROCESS

Software development normally follows a pattern called software development process model. This model is a structured process of software development that encompasses various stages of software development. A model constitutes various steps that lead to the building of efficient software. An important step in developing efficient software is to treat the entire development task as a process that can be controlled, measured, and improved. A process can be defined as a sequence of tasks that, when properly performed, produces the desired results (Humphrey, 1988). Software development process can be defined as a series of activities regarding a software product from such time the need to which the product is to respond is identified until such time the product is retired (Blum, 1994). In software, specifications have the tendency to change frequently after program code is written. However, most of the existing programming languages are too rigid when it comes to incorporating changes. There usually is a mix-up between the higher and the lower level constructs of the program so that one cannot be changed without affecting the other (Rumbaugh, 1996). Software development models solve this problem by permitting much of the code to be generated from a form that separates the high-level abstractions from the low-level programming details clearly, so that changes can be made to the implementation without affecting the logic (Rumbaugh, 1996).

Before discussing the different software development models, it is imperative to describe the characteristics that are inherent in all software process implementations. This will help in identifying the problems in the process that lead to imperfect software development (Blum, 1992). A simplified software development model can be divided into only two phases: decide what is needed and implement it (Blum, 1992). This implies that a software development model can be divided into two main phases, the first being the specification of requirements and the next being the development of the software to meet these requirements.

Quality assurance plays a vital role in the development of software. For any software to be developed and be useful, it has to be of a certain acceptable quality. The standards usually cited to be followed are ISO 9001 for design and development, and ISO 9000-3 guidelines for software. In software, the quality of the code determines the quality of the product. It depends on how the initial problem was attacked and what steps (model) were taken to solve the problem. The four major pointers to quality assurance are described below (Blum, 1992). These quality functions are defined in context of the whole process rather than in terms of the intermediate stages. Any software should follow these pointers in order to attain the expected quality mark.

Correspondence: This indicates how well the delivered system corresponds to the needs of the operational environment. The software is only as good as the requirements it satisfies. It is also important that the requirements are defined properly.

Correctness: This indicates the consistency of a product with respect to its specifications. The software built has to comply with the requirements and do what it is supposed to do. Hence, it is very important to build a software that fulfills the requirements.

Verification: This indicates whether or not the product is being built right. Care should be taken to see that the software that is being developed is done in the right manner, following the proper guidelines.

Validation: This indicates whether or not the product being built is the right one. It should be seen that the software built is not only working well but also that it is doing what it was build for.

Quality assurance ensures that the software development life-cycle used in developing a quality software product adheres to standards set forth for that product (Burch, 1995). It also includes monitoring of all software development stages from requirements stage to maintenance stage and taking preventive and corrective measures as and if necessary.

2.1.1 SOFTWARE DEVELOPMENT MODELS

Two major software development models cited in the literature are the waterfall model and the spiral model (Boehm, 1995). The classic life-cycle model is also cited in the literature. However, it is not as popular as the other two models mentioned since it is not in wide use anymore (Boehm, 1995). The waterfall and spiral models can be looked at as refinements of the classic life-cycle model. A software product cannot be successfully developed without the use of a software model. It is of critical importance that the developers select the right model while developing the software.

Specification of requirements can be said to be the most important phase of any software development process. It is the basis on which most of the software development projects are based. A clear cut requirements phase almost guarantees a good software and an unclear one almost always leads to disaster (Metzger, 1981).

Coding or writing the software in a machine readable language can be said to be the second most important stage in a software development process after the requirements specification stage. Software cannot be built without coding. Coding can be defined as a procedure to map models into actual programming code. The ultimate end of a software development project is to produce code.

The earliest model used for software development was the code-and-fix model (Boehm and Ross, 1989). This model depended completely on the coding stage for developing the software. Here, coding was done first and then the requirements, design, test and maintenance were thought of. This model led to an unstructured code, and the final product rarely met the user's specific requirements. It was very difficult to match program code to natural language requirements due to inadequate requirements specification phase. This led to its replacement with the more practical model of software development, the classic life-cycle model, which in turn led to the waterfall model and the spiral model. All the software development models have the following five stages common.

- *System engineering and analysis*
- *Design*
- *Coding*
- *Testing*
- *Maintenance*

We will discuss these stages before going in depth of the software development models.

System engineering and analysis: In this stage of the software development process, a preliminary investigation regarding whether there really exists a need for the software is

undertaken. This stage encompasses requirements gathering at the system level which includes the hardware and software parts with a small amount of top-level design and analysis. This stage may be called system/software requirements stage. This stage decides what is to be done or, what are the requirements that have to be fulfilled by the end product (Blum, 1992). In other words, it is the stage which defines the user requirements. The functional specifications for the software are given by the customer to the developer. The technical specifications are to be determined by the developer. The requirements are documented and reviewed with the customer. It also encompasses the feasibility study for both hardware and software requirements (Burch, 1995). Feasibility factors which refer to the likelihood of the software system being successfully developed and used are identified at this stage. Strategic factors relating to the software's support of business goals are determined. It also takes into consideration the cost and technical factors involved in the project. The development plan, scheduling and resources plan are also undertaken in this stage. A plan for the project is written and the technical problem is defined. During the problem definition activity, ideas about solutions will inevitably be discussed, but adoption of any specific programming solution is deferred until the design phase (Metzger, 1981).

Design: This stage translates the requirements into a representation of the software that can be assessed for quality before coding begins. This stage decides how to go about building the software so that it fulfills all the requirements (Blum, 1992). It also lays out details required to develop the software. This stage creates conceptual design alternatives and a description of the systems design concepts. Tools usually used to facilitate the design stage of software development are structure charts such as hierarchical diagrams, structured English which specifies detailed input and output, decision tables, decision trees, equations, data directories, Warnier-orr diagrams, and Jackson diagrams (Burch, 1995). The other commonly used software development tools are data-flow diagrams, control-flow diagrams and E-R diagrams. The design method, review procedures, and

technology plan are also a part of this stage. Either structured or object oriented approach may be used in the design phase of software development. In the structured approach, top-down method is most commonly cited in the literature. Here, usually a design document describing an acceptable solution to the problem is written. Usually many solutions are feasible, but one must pick the most suitable and go with it (Metzger, 1981).

Coding: The main objective of this stage is to convert software design applications into well documented software programs (Burch, 1995). The coding stage can be called the heart of the software development process. It is a very important stage in the software development process. This stage translates the design into machine-readable form. The language and structure to be used to develop the software are determined at this stage. One of the existing software development languages is usually used and the coding is usually done by software programmers in a modular form. Once the problem has been defined and a solution has been blue-printed, one can now build and test a program system according to that blueprint (Metzger, 1981).

One of the approaches that may be used in the coding stage is the use of fourth generation languages (4GL). 4GL is a term generally used to define proprietary, non-procedural languages developed to replace 3GLs such as COBOL (Burch, 1995). 4GLs use fewer lines of code to achieve what would require many lines in a 3GL (Burch, 1995). This approach calls upon one or more of the software development models to form a framework in which non-procedural languages, code-generators, database query systems, and other fourth generation approaches are applied. However, the use of fourth generation techniques still tend to be application-domain specific and it is still not being used to a large extent.

The 4GL enables the software developer to represent desired outputs in a manner that results in automatic generation of code to generate these outputs. Also, in this approach, the developer must conduct thorough testing, develop meaningful documentation, and perform all other transition activities. However, a drawback is that the requirements gathering phase requires that the requirements for the software be defined in a way that is understandable to a 4GL tool. This is not the case in reality since there is a lot of ambiguity in the actual requirements specifications.

Testing: This is the final stage of software development before its implementation. The purpose of software testing is reliability. Reliability requires error detection and removal. This stage focuses on the logical internals of the software, ensuring that all the statements have been tested. It also conducts tests to uncover errors and ensure that the defined input will produce actual results that agree with expected results. Software testing cannot turn a poorly designed software program into a good one, but it can help to determine the level of reliability before the software is released for use (Burch, 1995). After the programmers have built a product they are happy with, a separate group performs a new set of tests in as nearly a “live” environment as possible (Metzger, 1981).

Maintenance: Software maintenance is the modification of software after it has been delivered. Software maintenance begins as soon as the new software has been developed and converted to operations. All software systems are prone to change and these changes are made at this stage. Such stage reapplies each of the preceding development cycle steps to an existing program rather than a new one. Software maintenance begins after the installation of the product and lasts the lifetime of the product. There is usually a post-implementation audit done on the software by the customer regarding the status of the software.

Usually, the maintenance of existing software is given lower priority than the development of new software within an organization. If the software developed has to be modified on implementation, managers of non-system development departments will lose confidence in the abilities of developers. A serious consequence of this is the invisible application backlog. This is where disillusioned managers cease to submit some project development requests to the systems development department as they believe that they will never be developed. This will in turn lead to a shortage of innovative ideas. Studies indicate that organizations often have application development backlogs of between 2-4 years (Parker and Case, 1993). This indicates that after a request for a new system is made, the users may have to wait for many years before the software developers even start to work on the project.

2.1.1.1 THE WATERFALL MODEL

The waterfall model is the most frequently cited software development process model in the literature. It was developed by Boehm in 1976. It is a strong improvement on the classic life-cycle model in that it recognizes the feedback loops between the stages, and provides a guideline to confine the feedback loops to successive stages to minimize the expensive rework involved in feedback across many stages. This model (Figure 1) has become the basis for most software development projects today. It has also become the basis for most software acquisition standards in government and industry (Boehm, 1995). One of the reasons for the popularity of the waterfall model could be that it is easy to write contracts for it (Blum, 1992). Another reason could be that it accounts for almost all the stages that are usually followed in the software development process (Blum, 1994).

FIGURE 1: THE WATERFALL MODEL (*adapted from Blum, 1992*)

The disadvantage in this model however is that it is not very suitable for changing the requirements at a later stage in the development process (Boehm, 1995). The waterfall model is usually suitable for projects that have strong requirements specification stage and may prove to be useful when the software to be developed is structured as in the case of developing a Transaction Processing System (TPS). TPS supports the processing of a firm's business transactions and keeps an organization running smoothly by automating processing of the voluminous amounts of paperwork that must be handled daily (Parker and Case, 1993). The waterfall model may also be successfully used in developing complex applications and systems software.

However, using the waterfall model in cases such as building a Decision Support System (DSS) wherein intuition plays a very important role and the requirements are not very explicitly specified may prove to be a disadvantage. A DSS provides tools that enable managers to develop information in the manner that best suits the decisions that they are currently trying to make. DSS provides the manager with the computing and communication capabilities to develop his/her own decision models, data bases and report formats. DSS is a system that provides tools to managers to assist them in solving semi-structured and unstructured problems in their own, somewhat personalized, way (Parker and Case, 1993). This may lead to a breakdown in the waterfall model if used.

Expert Systems (ES) are software systems that imitate the reasoning processes of human experts and provide decision makers with the type of advice they would normally receive

from such human experts (Parker and Case, 1993). It can be said that ES are being developed to replace human experts. Building an ES with the waterfall model will most likely result in failure.

In the waterfall model, most of the aforementioned stages are common. However, the design stage is broken down into two parts, the preliminary design and the detailed design.

Preliminary design: In this stage, the details of the software product are chalked out and the feasibility of developing the product are defined and assessed. Also, the requirements and the facilities available to meet these requirements are assessed.

Detailed design: In this stage, the product is developed to the specifications. Detailed design can be commenced only after the completion of the preliminary design stage.

There are some limitations in using the waterfall model. The major one is that this model places emphasis on fully elaborated documents for early requirements and design phase (Boehm and Ross, 1989). Document-driven standards have pushed many project engineers to write elaborate specifications of poorly understood user interfaces followed by the design and development of large quantities of unusable code and thus end up pursuing stages in the wrong order (Boehm, 1995). In this model, one begins with a near complete understanding of the need (Blum, 1992) i.e., the model assumes that the requirements are obtained in one sitting with the customer. This is not always possible in reality with the

ever-increasing size and complexity of the software systems that need to be developed. Another shortcoming is the sequential method to software development advocated by the model. Moreover, in various classes of waterfall model applications, in which code is optimized for performance, modification of the code has become increasingly hard (Boehm, 1995). The waterfall model does not take into consideration the use or existence of prototypes which may indeed play an important role in reality. Also, it cannot be used successfully to build software wherein the requirements are semi-structured or unstructured. These shortcomings of the waterfall model have led to the formulation of the spiral model.

2.1.1.2 THE SPIRAL MODEL

This is an evolutionary approach to software development that incorporates the strengths of the above discussed model and resolves many of the difficulties. It is viewed as more of a refinement of the waterfall model through the years. The major feature of this model that distinguishes it from the waterfall model is that it creates a risk-driven approach to the software development process rather than the document or code driven approach of the waterfall model. The other major difference is that the spiral model incorporates prototypes into the development process while prototypes are visibly absent from the waterfall model. The major advantage of the spiral model is that its range of options accommodates the good features of the other model, while its risk-driven approach eliminates many of the difficulties such as taking into account the requirements at every

phase of software development. The spiral model defines an iterative process that incorporates four major components namely, management planning, risk analysis, engineering, and customer evaluation. These four phases (Figure 2) are applied iteratively, and with each iteration the software becomes more complete.

The four major activities involved in the spiral model are discussed below and they form the four quadrants of Figure 2.

Management Planning: This activity deals with the determination of objectives, alternatives, and constraints. It determines the requirements that the software has to comply with, the feasibility of the software to be developed, the different alternatives that can be used to develop the product and the limitations of these alternatives. Here, the objectives of the portion of the product being elaborated are identified (Boehm and Ross, 1989).

Risk Analysis: This activity deals with the analysis of alternatives and identification and/or resolution of risks. This is a kind of a cost/benefit analysis and there is a feasibility check done. The alternative means of implementing the portion of the product are identified (Boehm and Ross, 1989). One of the available alternatives is chosen depending on the constraints pertaining to the project.

FIGURE 2: THE SPIRAL MODEL (adapted from Boehm, 1995)

Engineering: This activity deals with the development of the higher-level product. Design of the product is done at this stage. Coding of the software is also done at this stage.

Customer Evaluation: This activity deals with the assessment of the results of engineering. Here, problems existing in the product developed are recognized and dealt with.

"The spiral model reflects the underlying concept that each cycle involves a progression that addresses the same sequence of steps, for each of its levels of elaboration, from an overall concept of operation document down to the coding of each individual program" (Boehm, 1995). Each cycle of the spiral begins with the identification of the objectives for the portion of the product being elaborated, the alternative means of implementing that portion of the product, and the constraints imposed on the application of these alternatives. The next step is to evaluate the alternatives relative to the objectives and constraints and identify and resolve any risks. Once the risks are evaluated, the next step is determined depending on the evaluation of the remaining risks. Then, a prototype is developed and if this prototype is operationally useful and robust enough, the subsequent steps would be the evolving series of evolutionary prototypes. These iterations continue until a satisfactory prototype which can lead to the final software product is developed. It is most likely that the spiral model may be used in primary parent firms rather than in the subsidiary firms due to lack of autonomy in the subsidiary firms.

In the spiral model, a software product is developed in a sequence of software development spirals, each loop around the spiral would correspond to one waterfall model iteration (Davis and Sitaram, 1994). Each such loop results in a new, improved and enhanced product, and the techniques used in developing the product on that loop are a function of the risk present at the time of development. However, there are some limitations in using the spiral model. First, there will be a high reliance on the expertise of the risk assessment for success. Second, this is a relatively new model and it may not yet be as widely accepted as the waterfall model and is not yet being extensively used (Boehm, 1995). The spiral model places a great deal of reliance on the ability of software developers to identify and manage sources of project risk (Boehm, 1995). The risk-driven specifications will be people-driven which can turn out to be a disadvantage.

The models of software development namely, the waterfall model and the spiral model have been discussed above. The waterfall model would be modified (as discussed in section 2.3) after thorough literature review and used as the basis of this study.

2.2 STRATEGIES TO REDUCE SOFTWARE DEVELOPMENT CYCLE TIME

As information technology becomes more and more accepted in society, people demand faster and better quality software. The time required to develop software plays a very important role in the final outcome of the software product developed. Software development time is the total time required for the development of the software from the requirements specification stage to the implementation stage. Software developers and their customers are often annoyed with the length of time it takes to develop a software. Software development activities (such as requirements specification, design, coding, and testing) usually take a great deal of time and resources. Product development time is very critical to product development success and consequently to the firm's survival. However, it seems almost inevitable that projects overrun planned completion dates (Vonk, 1990). Thus, in order to stay ahead in the competition, firms are adopting various strategies to reduce the software development time. A number of strategies that are used to reduce the software development time are discussed in the literature. This section discusses the software development time reducing strategies cited in the literature.

Application of Prototyping Approach: Prototyping is an approach that is most commonly used in the requirements specification and design stages, however, it is sometimes used in the other stages of the software development process too. The traditional modeling techniques for requirements definition are often not adequate for communication between developers and users, and mostly pay no attention to modeling the user interface of a

system (Vonk, 1990). Prototyping is an approach that uses working models as a means of communication between developers and users. Prototyping also supports and recognizes the iterative nature of learning by the users who on basis of prototype versions of the software, establish the real requirements for the system. This is an iterative approach wherein there is a heavy reliance on communication between the customer and the software developer. This approach of software development reduces the risk of an error possibly in the design or coding of the software going undetected till the final stages which can delay the project. The focus in this research is the time aspect of software development.

A prototype can be defined as an easily modifiable and extensible working model of a system or subsystem, not necessarily representative of a complete system, which provides users of the application system with a physical representation of key parts of the system before implementation (Boar, 1986 and Burch, 1995). A prototype may be used as a basis for a pilot and then later throw-away or it can be used in conjunction with the software development model to define user requirements and to expedite the software development process. The latter kind of prototype is called the evolutionary prototype. The requirements are defined through the creation of progressively more detailed models/prototypes of the software. The evolutionary prototype is used as a reference for the remainder of the project. A prototype also helps in identifying software requirements which when used with the waterfall model helps overcome the difficulty of having to specify the requirements in advance. Boar (1986) states that prototyping assumes that the

risk of poor requirements is best controlled and minimized by using a definition strategy which allows gradual learning and incremental change as part of the normal discovery process.

Application of Object-Oriented Approach (OOA): Object oriented software design results in a model that describes objects, classes and their relationships to one another (Burch, 1995). OOA programs can be reused and hence reduce the software development cycle time (Deitz, 1995). OOA is presently a significant factor in the development of general purpose software, with both IBM and Microsoft encouraging its use (Deitz, 1995).

The aim of object orientation is to bring the software industry in line with the hardware industry in which reusable components are used. This approach assumes that a new software can be developed from existing reusable software components. Here, there still are the requirements, design, code, and test stages. However, it differs from the other structured approaches in that coding is done by dividing the whole problem into parts called objects and then integrating the parts together. OOA can be used with any of the above discussed software development models. One of the major advantages of this approach over the approaches that consider the problem in its entirety is that object-orientation advocates the use of existing software. Here, an attempt is made to identify key classes and objects that are part of the problem domain. Once these classes and objects have been identified, the software engineer attempts to find them in a library of

existing, reusable classes and objects. If they can be found, the software is created by assembling new systems from existing components. If they are non-existent, then new components will have to be built. OOA simplifies the process for developing software, thus reducing the time to deliver a product to the market place, largely by reducing rework (Rauscher and Smith, 1995).

Use of Reusable Code and Application Packages: A good inventory of reusable code and designs help reduce the software development time (Phan, Vogel and Nunamaker, 1995). It also helps the developers to develop the product in parallel which can decrease the development time to a great extent. This strategy uses existing code to build new software.

Many programs require the same functions. These reusable modules can be stored in the library and they can be used in the future whenever required. When the activities to be supported by the software system are common to many organizations, there is a strong possibility that the problem can be solved by packaged software. Firms can use ready packages in order to reduce the time taken for software development.

Application of Incremental Development Approach: Software products should be managed in such a way that impacts of design changes on the development time are minimized. The use of the incremental approach in integration and testing of software allows versions to be compiled, integrated, and tested frequently (Phan, Vogel and

Nunamaker, 1995). Here, the above mentioned steps are performed at every stage in the software development process. Each higher level implementation depends on the functions provided by the earlier implementations (Blum, 1992). Testing at each stage will lead to early detection of problems and thus, reduce the time and cost in the development of the final product.

Use of CASE/Automated Tools: One of the most widely cited strategies to reduce the software development time is the use of Computer Aided Software Engineering (CASE)/automated tools. Here, automated tools are used to support the development tasks. These tools include code-generators, compilers, debuggers etc., which vastly reduce the time needed in software development. CASE's primary goal is to increase the productivity of software professionals and improve the quality of software produced while decreasing the time needed to develop them. These tools reduce the effort software developers devote to clerical tasks and/or assist software developers in intellectual tasks, thus improving time to market (Rauscher and Smith, 1995). No CASE vendor to date has been able to offer an integrated tool that can be used throughout the software development process (Burch, 1995). Use of a CASE system does not replace the fundamental business and systems planning, nor should it be used prior to the establishment of a structured software development methodology. A CASE tool does, however, provide a useful collection of tools, in a package, which can assist in some software development tasks (Burch, 1995). Examples of CASE tools are PacBase for main frames used for code generation, SignOn for AS/400 used for RPG generation,

Turbo Analyst for analysis, design on Windows platform, ADW on AS/400 for analysis, design, development, and planning, CSP for design development for main frames and easytrieve on main frames.

Use of Advanced Project Management Techniques: This strategy advocates the use of project management skills in order to reduce the software development time.

- **Using Scheduling Techniques:** Scheduling techniques, which include the Gantt Chart and the Program Evaluation and Review Technique (PERT) chart, help control timing and reduce the cost of systems projects (Burch, 1995). They give an indication of the approximate completion time and also the steps that can be undertaken to reduce the time.
- **Optimizing Work-Force:** Optimizing the use of the available work-force can also help reduce software development time. The software development project may optimize the work-force mix of critical and non-critical skills and should allow shifting employees from one functional area to another. There should not be a rigid work-force. The work-force may be changed so as to get the best results depending on the product that is being developed. According to Gupta and Wilemon (1990), being more creative and open to alternatives helps reduce the development time. According to the authors, developing a sense of urgency and responding to problems quickly goes a long way in reducing the development time.

- **Creating Team Motivation:** Having a motivated team working on the project may also help reduce the software development time. Rewards may be offered as an incentive to the better working of the team.
- **Using Work Breakdown Structure:** Breaking down the complete requirements into parts may also help in reducing the software development time. It is sometimes beneficial to establish a separate leader for successive software increments, particularly if the increments are large, with different technical centers (Boehm and Ross, 1989). By good planning, one can better understand and rationalize the software development process (Gupta and Wilemon, 1990). This in turn may help reduce the software development time. Plans aid coordination and communication, provide a basis for control, are often required to satisfy requirements, and help avoid problems (Rosenau and Lewin, 1984). Plans can be defined as a simulation of a project, comprising the written description of how the requirements will be satisfied. Plans can involve the defining of milestones very early in the project or the use of network diagrams to monitor the time schedule.
- **Focusing on Fault Avoidance:** Fault avoidance is also cited in the literature as a strategy that can be used to reduce the software development time. There is a rule of thumb in software engineering which states that if the modules are ordered according to the number of faults, the top 20% of the modules will contain 80% of the faults (Dalal, Horgan and Kettenring, 1994). Therefore, recognizing and correcting the faulty modules will greatly expedite software

development by reducing the risk of the finished software failing. Learning from the mistakes committed during the development of previous software also helps in expediting software development.

Use of Structured Methods and Techniques: To improve the quality of the system/software requirements phase and thus reduce the time taken to develop the software, the use of structured development methods and techniques such as software development models is advocated (Vonk, 1990). Most of the software development problems arise from incorrect or incomplete requirements definition which can be overcome by following a structured method of software development.

Verification of Requirements: It is very important to keep verifying at every stage of software development whether or not the software is being built according to the requirements. If the requirements are not being fulfilled, the development of the software is bound to be unsuccessful.

2.3 A *PRIORI* MODEL

The waterfall model discussed earlier was modified appropriately in order to overcome some of the difficulties such as the need for complete understanding of the requirements before beginning the development process, and the impractical sequential approach advocated by the model. This modified waterfall model (Figure 3) was used for the research and was shown to the respondents during the interview.

The following three changes have been made to the existing waterfall model in order to come up with the *a priori* model. These changes are based on the literature review undertaken and the four pretests conducted for the purpose of this study.

First, a major disadvantage cited for the waterfall model was that it requires complete specification of the requirements before the development process can be undertaken. The alternative can be to have constant communication among the stages of software development. In order to represent this communication from each of the other stages to the requirements stage, feedback loops are added to the waterfall model. These feedback loops allow the development process to take into consideration any changes in the requirements.

FIGURE 3: A PRIORI MODEL

The second modification made is the addition of a prototype life-cycle to the stages of the model. According to the literature studied, prototyping was found to be vastly used in the development of software. Thus, the model will take into consideration the possibility of prototyping at every stage of software development.

Finally, quality assurance is an integral part of the software development life-cycle. According to the pretests conducted, it plays an important role throughout the software development process. This has been taken into account in the *a priori* model.

In summary, the sections above have reviewed the various software development process models, strategies that may prove helpful in reducing the software development time and finally proposed an *a priori* model of the software development process that was used for the research.

3 RESEARCH OBJECTIVES

There were two research objectives for this study:

1. To understand and analyze the software development process being used in high technology firms. An *a priori* model¹ (Figure 3) was used in order to achieve this objective.
2. To determine the effectiveness of the strategies that are being used to expedite software development.

¹As described in Section 2.3, an *a priori* model was developed based on the literature review and pretests conducted. It was used for framing the questionnaire for data collection.

4 METHODOLOGY

This section is concerned with the steps undertaken to conduct the study. It is divided into five sub-sections. Section 4.1 discusses the research design and strategy. Section 4.2 describes the sample selection procedure. Section 4.3 and 4.4 describes the field work and preliminary investigation conducted. Finally, Section 4.5 provides a detailed account of the data collection and analysis.

4.1 RESEARCH DESIGN AND STRATEGY

The research design used in this study was an **ex-post facto design**. None of the variables were manipulated in the study. It was a field study within the respondent's environment. The research strategy used was **exploratory study of software development**. The exploratory study involved personal interviews with a project member responsible for new software projects. An exploratory study was selected since not much empirical research was done in the proposed area. Using the results of the study, two general models for software development were developed and the strategies that are perceived to be effective in reducing the software development time were determined.

4.2 SAMPLE SIZE AND SELECTION

A total of 22 small and medium sized high technology firms from Ottawa region in Canada and Bangalore region in India² were considered for the purpose of the study. Two subjects were interviewed in one of the Canadian firms, and one each in the remaining 21 firms. A total of 23 subjects were interviewed, of these, 14 were from Canadian firms and 9 were from Indian firms. The convenience sample was selected from a population consisting of software development units in high technology firms. Only firms that had been involved in some new software product development in the past couple of years were selected for the purpose of the study.

Software development projects can be internal or external depending on whether the project was undertaken to develop software for use within the firm or for the external market. Software can also be categorized as systems software and applications software. Applications software includes user-oriented programs written in programming languages. It includes the programs, software packages, and tools needed for such specific, end-user oriented tasks as billing, accounts receivable, word processing, payroll, database management, and executive support (Parker and Case, 1993). Applications software can also include complex software like embedded software, and robotics. Systems software consist of 'background' programs that enable applications software to run smoothly on a

²India has been selected as part of the study because it is developing into one of the top most Software developers in the world. India is home to many national and multinational high technology firms.

specific set of hardware (Parker and Case, 1993). This study focused on software projects that involved the development of systems software or complex applications software like software in robotics, and switching systems which are embedded software. The study did not look into TPS projects such as software for the working of credit cards and very simple software projects.

4.3 FIELD WORK

The selected companies were contacted through telephone and the receptionist was asked to direct the call to a project manager (or the R&D manager) responsible for a recently completed software project. It usually snowballed from one manager to another before the person who could be interviewed was found. All these persons were explained the purpose of the study and the firm selection criteria. In case the firm met the selection criteria, the first available project member who qualified to answer the questions of this study was asked for an interview.

4.4 PRELIMINARY INVESTIGATION

The researcher pre-tested the questionnaire by interviewing 2 subjects in Ottawa, Canada and 2 in Bangalore, India³. On the basis of these pre-tests, suitable modifications were made in the questionnaires and the *a priori* model.

³These tests were conducted during the researcher's trip to India during August 1996.

4.5 DATA COLLECTION AND ANALYSIS

A **Primary Data Collection** method was used wherein personal interviews were conducted with a project member involved in new software development projects. Moreover, back-up documentation such as written descriptions, field reports and flow-charts were requested.

At the beginning of the interview, a letter from the thesis supervisor, as shown in Appendix I, explaining the purpose of the study and ensuring confidentiality of the information was presented to the respondent. Interview probes were used to supplement the interviews. A questionnaire was used after the interview to investigate the strategies followed by the firm to reduce the software development time. The strategies being used to reduce the time for software development were found through an exhaustive literature search and were used to prepare the questionnaire. The respondents were told that the results would be in an aggregated form and no firms would be named in the results. Moreover, the findings of this study were made available to the participating firms if they so desired.

There were two purposes for this study. The first purpose required the understanding of the software development process followed in the high technology industry. In order to do this, the respondent was requested to select a recent (last 2-3 years) software product that

he/she was responsible for developing. The respondent was then asked through a set of open ended questions to describe the software development process followed in the development of the selected software product. The *a priori* model (Figure 3 without any links) of software development process was shown to the respondents in order to help them keep the discussion in focus and not miss any of the stages due to oversight. The interview was conducted using a probe (Appendix II: Questionnaire A). The analysis to understand the software development process, was qualitative in nature. The interviews were transcribed and an in-depth analysis was conducted as explained below. Since the answers were not very specific, the questions asked by the researcher and the researcher's interpretations of the answers played a vital role. To represent the qualitative data obtained in the form of a model for software development, the approach adopted was based on the principles of matrix thinking and inductive research. The idea being to reduce lengthy verbal responses to essentials and then grouping them into well-defined categories. The responses obtained were then categorized based on the number of respondents giving similar responses. Each set of similar responses were then presented in the form of a software development model representing the software development process followed in high technology industry. This study resulted in two software development models because of the differences found in the processes followed in Canada and India.

The second purpose of this study was to determine the effectiveness of the strategies adopted by these firms to reduce the software development time. After the interview, the respondents were asked to fill out a questionnaire (Appendix II: Questionnaire B)

regarding the effectiveness of the strategies used by them to reduce the software development time. The questionnaire listed from the literature all the strategies that help reduce software development time. The respondents were asked to rate them on a 5-point itemized rating scale. The strategies were then rated according to the average points it got from the responses of all the participating firms. T-tests were conducted to find if the results were significant. This helped determine the effectiveness of each strategy as perceived by high technology firms. The respondents were also asked to list any other strategies that they might have used. These strategies have been incorporated in the results. Both, the personal interview with the respondent, and the administration of the questionnaire were conducted in one meeting and the time taken for each meeting was approximately one to one and a half hours.

DISCUSSION OF RESULTS

5. SOFTWARE DEVELOPMENT MODEL

In this section, two models that reflect the software development process in high technology firms are proposed. It will explain in detail the various stages and linkages found among the stages of the software development process.

5.1 SOFTWARE DEVELOPMENT PROCESS MODEL

The first research objective, i.e., to understand and analyze the software development process model in high technology firms will be discussed in this section. During the interviews conducted, a pattern of differences was observed in the software development processes being followed in respondent firms. Hence, two models of software development have resulted from this study.

The data for this objective was collected through a series of open-ended questions. A project member (the R&D Director in most cases), actively involved in the software development project was interviewed and the necessary information collected. To represent this qualitative data on the software development process in the form of a model, the approach adopted was based on the principles of matrix thinking and inductive research; the idea being to reduce lengthy verbal responses to essentials and then grouping them into well-defined categories. Patton (1981) has discussed the use of matrix thinking to organize, arrange, and analyze qualitative data. According to him, the power of matrix

thinking resides in the simplicity of the tables created. Further, he has highlighted that matrix thinking is common in social sciences generally and evaluation specifically. Mintzberg's (1979) discussion on inductive research is also very relevant to this study. He has compared inductive research to detective work, where one tracks down patterns and consistencies. The study of Mintzberg, Raisinghani, and Théorêt (1976) on model development, provided insight into analyzing qualitative data. This study follows a similar methodology.

Section 5.1.1 gives a detailed description of the stages followed in the software development model. Section 5.1.2 describes the linkages among the various stages, and finally, Section 5.1.3 explains the software development models in Canada and India.

5.1.1 DETAILED DESCRIPTION OF THE STAGES IN THE MODEL

Stages of the software development process model and the activities that are undertaken in these stages are discussed below.

Table 1 summarizes the types of projects that were studied. It was observed that for all the systems software studied, the requirements were unstructured. Whereas, for complex applications software, 78% of the projects had structured requirements. Transcribing the interviews resulted in no significant differences in the models based on the type of project. Hence, the data analysis conducted did not differentiate the models on this basis.

TABLE 1: TYPES OF SOFTWARE PROJECTS STUDIED

Projects	Structured applications software	Structured systems software	Unstructured applications software	Unstructured systems software	Total
External	13	0	3	4	20
Internal	1	0	1	1	3
Total	14	0	4	5	23

Out of the 23 software projects studied, one was found to be not using any formal model of software development. The software was developed from instinct. It was found that usually a week was spent preparing features that had to be present in the software. It was claimed that it was feasible to state software requirements only if they were very clear-cut, which is not the case in reality. The response obtained from this interviewee will not be included in the analysis. Therefore, the data analysis conducted considered the data collected from the remaining 22 respondents.

The two major motivations for the development of new software were found to be either a customer's request for the software, or the firm's own requirement/intuition to develop the software. Responses obtained from Canadian and Indian firms were similar. One of the respondents said that the software was developed with the intuition that it would help software technology. He claimed that his idea went against the existing technological

beliefs and yet the software ended up being a commercial success. It was found that when the idea to develop the software was initiated by the firm's R&D department, a prototype was first built and shown to the customer and the final product was developed only after the customer/end-user approved it or agreed that it would be a feasible product. It was also observed that 50% of the applications software projects were customer driven and that 80% of systems software projects were driven by the firm's R&D department. This could be because applications software is more for end-users while systems software is used to run the applications software. Respondents said that the situation of competitors and their products in the market played an important role in the starting up of a new software project. One interviewee went as far as to say that they used the competitor's products as prototypes and then improved on these products in order to come up with their version. Once it was decided to undertake the project, the software development process was started.

This study resulted in the software development model being followed in the high technology firms. The six stages existing in the model are discussed below. The interviews were transcribed and no significant differences were observed in the stages followed by Canadian and Indian firms. All the stages were common for the models followed in both the countries. Hence, the section will be the same for both Canadian and Indian firms.

System/Software Requirements Stage

The system and software requirements stages that were kept separate in the *a priori* model were combined as it was found that in high technology firms the system and software requirements were conducted in a single stage of the software development process. Literature (Boehm, 1995) claimed that the systems and software requirements were conducted in two separate stages. However, this study showed that the high technology firms conducted both systems and software requirements in a single stage. It was claimed that the systems and software requirements are interdependent and so, conducting both the activities at the same time saves resources. The linkages to this stage were adjusted appropriately and the irrelevant linkages were eliminated. All respondents unanimously agreed that there existed a system/software requirements phase in the software development process. They claimed that it was a very important stage of software development and this was where the foundation of the software was laid. They agreed that the requirements phase was a key determining factor in the success or failure of the software. This fact is supported by existing research (Burch, 1995). If the requirements are not clear and specific, the end product usually does not meet the demand. The business department played a very important part at this stage. The technical staff, though involved, did not play a vital role in the process at this stage.

All respondents agreed that the specifications/requirements were first mapped out and only then was the actual development of the software undertaken. Feasibility tests were

conducted to determine whether or not the software project at hand was feasible in terms of finances, technology, and other resources. One interviewee claimed that financial staff looked into the functional specifications of the software since the software developed by them was meant for financial institutions. The overall architecture of the software was determined at this stage. It was also found that the customers (in the case of customer-driven software) were shown a prototype at this stage and only after their approval was the actual project undertaken. Customers usually did not know what they exactly wanted, so, it was very important to specify the requirements as best as they could in order to develop efficient software. One respondent can be quoted as saying “the main objective at this stage was to get a right spectrum of requirements. The key thing at this stage was to have the right people playing the right role and make decisions by consensus”.

Preliminary Design Stage

All respondents agreed that they had a design phase in the software development process, although there were five respondents who claimed that explicit distinction was not made between preliminary and detailed design. The overall design of the software was performed at this stage. One respondent called this the “technical design” phase. The technical staff mapped out the design details and it went through an approval procedure. At this stage, the specifications were checked and the implementation aspects of the software were studied. This stage usually resulted in a design document developed by the

technical staff and approved by the business staff. This stage broke down the software requirements to the detailed level so that the software could be coded.

Detailed Design Stage

Respondents claimed that sometimes there was no clear boundary between the preliminary and detailed design stages even though both stages were present. At this stage, a design document was developed. This document was used for the actual development of the software. Software decisions on languages and databases to be used in developing the software were also taken at this stage.

Code and Debug Stage

There was unanimous agreement that Code and Debug was a phase present in the software development process. The coding stage was where the actual development of the software was undertaken. All aspects of software development were handed over to the technical staff. Coding was done using the language and databases stated in the design documents and could be either modular or not, depending on the complexity of the project. This study has not dealt with the technical aspects of coding.

This was the stage where the quality of the software developed was determined. Software developers were required to follow company guidelines and standards while coding the

software. These guidelines usually pertained to documentation, structure, modularity and clarity of the software. Respondents said that the developers were usually trained in the beginning and then it was entirely upto them to develop the software according to the specified standards. This stage was said to be the most time consuming one in the development process.

Test and Pre-Operations Stage

All the respondents agreed that the software developed was tested and pre-operations were conducted on the software in the developer's environment before it was sent out in the market. 68% of the respondents claimed that the software was tested after each stage of the development process. The rest said that they tested the software only after it was developed completely. However, all the respondents said that the software was tested once development was complete. The term they used for this final product testing was "integral testing". Once the program/software was developed, the programmer conducted unit tests on the databases/code. Then, it was handed over to the Quality Assurance (QA) people. They had a detailed idea of the specifications and they tested the software accordingly.

One respondent said they used a system of "Buddy Testing" wherein the documented software was handed for inspection to a person new to the project, as they found it was not a good strategy to test their own software. One respondent was quoted as saying "we

found that the developers were good at catching each other's bugs". Usually developers conducted tests till 80% of the software was complete and improved on it. Only after the developers were satisfied did the QA department take over and conduct tests. Testing was done by the QA department and the software went back to the developers for fixing of bugs. After the bugs were fixed, the product was again tested by the QA department. This loop continued till the tester was satisfied with the result.

Operations and Maintenance Stage

All the respondents were unanimous that the products were maintained after they were developed and installed. Most of the firms were under a contract to maintain the software for a period of time, usually one to two years for applications software. There was also a strong possibility of a change request from the user after the software had been installed, and this request was complied with. Firms had a hot-line support facility wherein they fixed the bugs in the software immediately after the users pointed them out. It was also mentioned that if the bug was not a major one, then it would be taken up as an enhancement and would be worked upon in the next version of the software.

5.1.2 FORWARD AND FEEDBACK LINKAGES AMONG THE STAGES

The preceding section discussed the various stages followed in a software development project. This section will discuss the linkages existing among these stages.

Transcribing the interviews and using Table 2, the following linkages were determined to be present in all projects studied.

- System/Software Requirements to Preliminary Design
- Preliminary Design to Detailed Design
- Detailed Design to Code and Debug
- Code and Debug to Test and Pre-operations
- Test and Pre-operations to Operations and Maintenance

The system/software requirements stage is the system engineering and analysis stage described in the literature review section of this study. Similarly, design, code, and test stages have been described in detail earlier. The claim in the literature that a linear link existed among all the stages of the software development model was affirmed by this study.

Table 3 shows the percentage of respondents who claimed the presence of the linkages. The links that were used by 40% or more firms were considered to be important and were used to develop the models. The links that were used by less than 40% were not included in the model.

All respondents in the Canadian and Indian firms agreed that quality assurance played a vital role throughout the development process. The importance of quality assurance in software was summed up by a respondent who said “We have a standard set of guidelines built into the process and there is no choice but to follow it”.

Tables 2 and 3 were used to determine the software development model used in the high technology firms. Section 5.1.3 discusses the software development models followed in Canadian and Indian high technology firms. Explanation of Linkages used in Table 2:

LA - Forward link from system/software requirements to preliminary design.

LB - Forward link from preliminary design to detailed design.

LC - Forward link from detailed design to code and debug.

LD - Forward link from code and debug to test and pre-operations.

LE - Forward link from test and pre-operations to operations and maintenance.

LF - Feedback link from detailed design to preliminary design.

LG - Feedback link from code and debug to detailed design.

LH - Feedback link from test and pre-operations to code and debug.

LI - Feedback link from operations and maintenance to test and pre-operations.

LJ - Feedback link from preliminary design to system/software requirements.

LK - Feedback link from detailed design to system/software requirements.

LL - Feedback link from code and debug to system/software requirements.

LM - Feedback link from test and pre-operations to system/software requirements.

LN - Feedback link from operations and maintenance to system/software requirements.

LO - Use of prototypes in system/software requirements stage.

LP - Use of prototypes in preliminary design stage.

LQ - Use of prototypes in detailed design stage.

LR - Use of prototypes in code and debug stage.

	LA	LB	LC	LD	LE	LF	LG	LH	LI	LJ	LK	LL	LM	LN	LO	LP	LQ	LR
F1	√	√	√	√	√	√	√	√	√	√	√	√	√			√		√
F2	√	√	√	√	√	√	√	√					√			√	√	
F3	√	√	√	√	√	√	√	√	√	√				√	√	√		√
F4	√	√	√	√	√	√	√	√		√	√	√	√	√				
F5	√	√	√	√	√			√	√	√					√	√		√
F6	√	√	√	√	√	√	√	√							√		√	√
F7	√	√	√	√	√	√	√	√	√	√					√	√	√	√
F8	√	√	√	√	√		√						√		√	√		√
F9	√	√	√	√	√	√	√	√	√	√	√				√			
F10	√	√	√	√	√	√	√	√	√	√	√	√	√					
F11	√	√	√	√	√	√	√	√	√	√	√	√	√					
F12	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√			
F13	√	√	√	√	√	√	√	√	√									
F14	√	√	√	√	√	√	√	√			√					√		√
F15	√	√	√	√	√	√		√		√	√	√	√	√	√	√	√	√
F16	√	√	√	√	√	√		√		√								
F17	√	√	√	√	√		√	√	√		√	√		√	√	√	√	
F18	√	√	√	√	√	√	√				√	√	√	√	√			
F19	√	√	√	√	√	√	√	√	√	√				√	√	√	√	
F20	√	√	√	√	√	√	√	√			√		√	√	√	√	√	√
F21	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	
F22	√	√	√	√	√	√	√	√	√	√	√	√	√	√				

TABLE 2: THE LINKAGES EXISTING BETWEEN THE STAGES OF THE MODEL

√ - Indicates the presence of the linkage.

F1 - F9 Represents Responses from Subjects interviewed in India.

F10 - F22 Represents Responses from Subjects interviewed in Canada.

LA - LR - Linkages described above.

NO.	PARTICULARS OF LINKAGES	% YES RESPONSES	
		CANADA	INDIA
	FEEDBACK LINKS		
1	Detailed design to preliminary design	92%	78%
2	Code and debug to detailed design	85%	89%
3	Test and pre-operations to code and debug	92%	89%
4	Operations and maintenance to test and pre-operations	62%	56%
5	Preliminary design to system/software requirements	62%	67%
6	Detailed design to system/software requirements	77%	33%
7	Code and debug to system/software requirements	62%	22%
8	Test and pre-operations to system/software requirements	62%	44%
9	Operations and maintenance to system/software requirements	62%	22%
	PROTOTYPES		
10	Use in system/software requirements stage	54%	67%
11	Use in preliminary design stage	46%	67%
12	Use in detailed design stage	38%	33%
13	Use in code and debug stage	23%	67%

TABLE 3 : PRESENCE OF LINKAGES AMONG VARIOUS STAGES OF MODEL

FIGURE 4: SOFTWARE DEVELOPMENT MODEL FOLLOWED IN CANADA

FIGURE 5: SOFTWARE DEVELOPMENT MODEL FOLLOWED IN INDIA

5.1.3 SOFTWARE DEVELOPMENT MODELS IN CANADA AND INDIA

This study based on questions and probes developed in Questionnaire A (Appendix II) helped in generating models of the software development process for the high technology firms in Canada and India. Based on Tables 2 and 3, the models were analyzed. Figures 4 and 5 show the software development models followed in Canadian and Indian high technology firms. However, two of the Canadian respondents claimed that they used the Spiral model which is a series of iterations of the above model (Figure 4), implemented back-to-back. This model can be equated to developing the software in increments and each increment consisting of the above model. This section highlights some observations associated with the software development process.

It was observed that in most cases the software requirements were determined by having discussions with the customer/end-user. The customer played an important role in the specification of requirements.

The most significant difference found in the software development models followed in Canada and India was the greater use of prototypes by the Indian companies. This could be due to the fact that most of the Indian companies studied were multinational companies and it can be hypothesized that they did not have much freedom to go ahead with the development process until they had approval from the parent company. However, most of

the Canadian companies had autonomy and they could go ahead and start off with the development process. Another reason could be the lower cost of labor in India. Prototyping is a people-intensive approach and the cost of labor may play an important role in the extent of use of prototypes. The other reason found to be was that the market in Canada is much more competitive and so the time taken to bring the software in the market plays a vital role in the commercial success of the software. Prototyping helps reduce software development time in large projects. However, it can lead to delays in smaller projects. The other significant difference was the absence of feedback from the various stages to the requirements stage in the model followed in India. It can be hypothesized that the requirements specification was very rigid in the Indian firms and no major changes could be incorporated after the requirements were specified at the start of the project.

The model followed in the Canadian firms (Figure 4) had almost all the linkages that were in the *a priori* model used for the study. The only linkages missing were the absence of prototypes in the detailed design, and code and debug stages. Similarly, the only links missing from the model followed by Indian firms (Figure 5) were feedback from detailed design, code and debug, and maintenance to the system/software requirements stage and absence of prototypes at detailed design stage.

Surprisingly, not much has been mentioned in the literature regarding the role of quality assurance (QA) in the software development process. This study found that QA played a very significant role during the development process. The QA department was reported to

have been active right from the requirements phase and was also said to be involved in almost all meetings and reviews.

Respondents claimed that most of the QA activities were conducted informally and most of them agreed that there was room for improvement and they were always looking for ways to improve, and they usually had a “lessons learnt” session after the project.

The respondents said that checks were conducted to verify that what the customers have is what they needed. Correspondence was usually checked through feedback from customers, users, and sales representatives. One respondent can be quoted as saying “Correspondence was on the forefront of our minds at all times in the development process”. Correctness i.e., checking if the product was consistent with the requirements was claimed to be a necessary requirement for development. However, one respondent said that this check was not conducted as the requirements keep changing. Verifying if the software was being built in the right manner, with proper guidelines was said to have been performed. Firms had certain guidelines regarding documentation, structure, modularity, and clarity of the software. Respondents said that they sometimes had consultants come in to do development reviews. Validation of the software was said to be part of the testing process and was usually done informally. It was important because the product goes directly to the end-user and if it does not function as expected, the firm would get to hear complaints from the users.

However, the respondents confessed that not enough checks/reviews were conducted to inspect if the design and code followed software standards. Modularity reviews were conducted to discuss about the functionality to be built. They said that the developers were trusted to come up with good products and in most cases formal checks were not conducted since they were time consuming. The firms followed their own company standards for software development and standards such as ISO (e.g. ISO 9001), and SEI (e.g. CMM 1.0) were not being implemented even though they existed. However, company standards were developed based on these existing standards. Respondents said that software technology is fast growing and sometimes the standards in use are the ones developed for the older technologies and are often outdated.

6. STRATEGIES USED TO REDUCE SOFTWARE DEVELOPMENT TIME

The second objective of this study was to determine the strategies high technology firms find effective in reducing the software development time. The data analysis for this objective was quantitative and was performed with the help of the statistical package, SPSS.

Strategies were identified from the literature as being helpful in reducing the software development time. The respondents were asked to rate the effectiveness of these strategies on a 5-point itemized rating scale with 1 representing not effective and 5 representing very effective. Respondents marked N/A for not applicable/not used.

Based on similarity, these strategies were grouped into eight broad categories. Following is the list of these categories:

- I. Application of Prototyping Approach
- II. Application of Object-Oriented Approach
- III. Use of Reusable Code and Application Packages
- IV. Use of Incremental Development Approach
- V. Use of CASE/Automated Tools
- VI. Use of Advanced Project Management Techniques
- VII. Use of Structured Methods and Techniques

VIII. Verification of Requirements

Before presenting the results, it is necessary to mention that this was an exploratory study with a limited sample size that restricted the performance of any rigorous statistical analysis on the data collected. However, best efforts have been made to derive maximum meaning out of the data collected. Still, the results have to be treated with caution. This study involved data collection in Canada and India. The data analysis was done separately for the two countries.

General observations are presented in the form of descriptive statistics. Descriptive statistics refers to analysis and synthesis of data so that a better description of the situation can be made and thereby promote a better understanding of facts. One should note that it does not attempt to select the best alternative, but only describes the choices that are open to the decision maker (Srivastava, Shenoy and Sharma, 1983). The descriptive statistics results are shown in the form of Table 4 and Figure 6. Table 4 presents the summary of mean effectiveness for each of the strategies under the eight categories for both Canada and India and results of t-tests performed to check if they were statistically significant at significance levels of 0.01, 0.05 and 0.10. Statistical importance was determined by performing t-test on variables having a mean over 3, which represents the center point on the 5-point scale. T-test was also performed to determine whether the strategy means were significantly different from the overall means of 3.65 and 3.93 obtained for the strategies in Canada and India. However, the descriptive analysis below will only consider

the t-test results conducted with 3 as the reference. Descriptive analysis was done only for the t-tests conducted with 3 as the reference because it is the mid-point on the 5-point scale used for the study. The results of the t-test with the overall strategy means are shown in Table 4. Figure 6 shows the mean effectiveness of the strategies as perceived by the Canadian and Indian firms in the form of a graph.

The responses that indicated non-applicable (not used) were not considered for the analysis because the fact that they were not used indicates that the responses will not indicate the effectiveness. Hence, they were eliminated from the statistical analysis. Respondents were also asked to indicate the percentage of projects in which they used these strategies during the past two years and this data was used in the descriptive statistics. Table 5 presents the percentage of projects that used the strategies during the past two years.

Each of these strategies are explained below. Mean effectiveness of the strategies are indicated and these values are for Canada and India respectively. It is also indicated whether the results were significantly different from 3.

I. Application of Prototyping Approach

The importance of using Prototypes in the software development process has been highlighted extensively in the literature. Table 4 presents the summary of the results

obtained regarding the effectiveness of prototypes in expediting software development in Canada and India.

The study showed that 50% and 60% of software projects in Canada and India respectively used prototyping. Of these, 80% projects used evolutionary prototypes and 30% used throwaway prototypes in Canada; 70% used evolutionary prototypes and 50% used throwaway prototypes in India.

Prototyping in the requirements stage (mean = 4.00 and 3.80) is perceived to be significantly more than moderately effective at α level 0.01 and 0.1 respectively in expediting software development. Prototyping at the design stage (Mean = 4.10 and 4.00) was considered to be significantly effective at α level 0.05, slightly more than at the requirements specification stage. However, prototyping at the coding stage was not considered to be even moderately effective in expediting software development (mean = 2.71 and 2.89). However, this result was not significant.

One respondent said that prototyping was “popular in new projects using a new technology which was on the market for a short period of time”. However, two respondents said that prototyping increased the development time because time which was spent in prototyping could be used in developing the software. It can be hypothesized that this is true in smaller projects. However, in larger projects, the risk of developing the actual

NO.	STRATEGIES	MEAN		MEAN+	
		CANADA	INDIA	CANADA	INDIA
I	Application of Prototyping Approach				
1	Prototyping in Requirements Stage	4.00***	3.80*	4.00*	3.80
2	Prototyping in Design Stage	4.10**	4.00**	4.10	4.00
3	Prototyping in Coding Stage	2.71	2.89	2.71	2.89
II	Application of Object-Oriented Approach				
4	Segmentation of Software	4.00***	4.50***	4.00	4.50***
5	Divide software into objects	3.63	3.90***	3.63	3.90
III	Use of Reusable Code and Application Packages				
6	Use Existing code from own inventory	4.30***	3.80*	4.30***	3.80
7	Use code available in the market	3.92**	3.38	3.92	3.38
IV	Use of Incremental Development Approach				
8	Incremental development	3.92**	4.60***	3.92	4.60***
V	Use of CASE/Automated tools				
9	CASE/Automated tools in requirements stage	3.17	3.67**	3.17	3.67
10	CASE/Automated tools in design stage	2.67	4.11***	2.67	4.11
11	CASE/Automated tools in coding stage	2.71	4.00*	2.71	4.00
VI	Use of Advanced Project Management Techniques				
12	Use of scheduling charts	3.82**	4.00***	3.82	4.00
13	Use of software schedule tracking tools	3.82**	3.80*	3.82	3.80
14	Shift employees	3.64**	3.10	3.64	3.10
15	Mix technical and non-skilled staff	3.09	3.00	3.09	3.00
16	Be innovative	4.43***	4.40***	4.43***	4.40
17	Seek new ideas	4.50***	4.40***	4.50***	4.40

TABLE 4: MEAN EFFECTIVENESS OF STRATEGIES

NO.	STRATEGIES	MEAN		MEAN+	
		CANADA	INDIA	CANADA	INDIA
VI	Use of Advanced Project Management Techniques				
18	Create a sense of urgency	4.00***	3.10	4.00	3.10
19	Respond immediately to problems	3.93**	4.70***	3.93	4.70***
20	Motivate team with rewards	2.92	4.70***	2.92	4.70***
21	Use of different leaders	2.20	2.50	2.20	2.50
22	Check for faults	4.29***	4.40***	4.29***	4.40***
23	Define milestones	4.43***	4.30***	4.43***	4.30
24	Learn from mistakes	4.21***	3.90***	4.21***	3.90
VII	Use of Structured Methods and Techniques				
25	Use of structured model (theoretical)	3.09	4.10***	3.09	4.10
26	Use of structured model (firm's own model)	2.80	4.44***	2.80	4.44***
VIII	Verification of Requirements				
27	Verification of requirements	4.21***	4.50***	4.21***	4.50***

TABLE 4 (cont'd): MEAN EFFECTIVENESS OF STRATEGIES

*** Significantly different from 3 at the 0.01 α level

** Significantly different from 3 at the 0.05 α level

* Significantly different from 3 at the 0.10 α level

+ Indicates results of t-test conducted with the overall means of 3.65 and 3.93 obtained for strategies in Canada and India as reference.

STRATEGY	% OF USE	
	CANADA	INDIA
Application of Prototyping Approach	50	60
Application of Object-Oriented Approach	50	50
Use of Reusable Code and Application Packages	60	50
Use of Incremental Development Approach	70	75
Use of CASE/Automated Tools	25	40
Use of Advanced Project management Techniques	80	65
Use of Structured Methods and Techniques	50	75

TABLE 5: USE OF VARIOUS STRATEGIES

FIGURE 6: MEAN EFFECTIVENESS OF STRATEGIES

X - Axis indicates the 27 strategies labelled a to aa

Y - Axis indicates the mean effectiveness

Strategy	Description		
a	Prototyping in requirements stage	n	Shift employees
b	Prototyping in design stage	o	Mix technical and non-skilled staff
c	Prototyping in coding stage	p	Be innovative
d	Segmentation of software	q	Seek new ideas
e	Divide software into objects	r	Create a sense of urgency
f	Use existing code from own inventory	s	Respond immediately to problems
g	Use code available in the market	t	Motivate team with rewards
h	Incremental development	u	Use of different leaders
i	CASE/Automated tools in requirements stage	v	Check for faults
j	CASE/Automated tools in design stage	w	Define milestones
k	CASE/Automated tools in coding stage	x	Learn from mistakes
l	Use of scheduling charts	y	Use of structured model (Theoretical)
m	Use of schedule tracking tools	z	Use of structured model (firm's own)
		aa	Verification of requirements

product and then realizing a mistake far outweighs the time spent on prototyping in the case of large projects.

Thus, based on the strategies mentioned in the questionnaire and the responses, it can be concluded that prototyping is a key strategy that can have a definite impact on reducing the software development time.

II. Application of Object-Oriented Approach

Object-Oriented approach has been given great emphasis in the literature for expediting software development. The summary of findings regarding the effectiveness of these strategies is shown in table 4.

50% of software projects in Canada and India used the object-oriented approach. This approach has two components: Segmentation of software and, dividing the software into objects. The strategy to segment the software requirements into various sections and then integrate them obtained a mean value of 4.00 and 4.50 (significantly different from 3 at α level 0.01) which indicates that it was perceived to be a very effective approach. However, dividing the software into objects and then joining them together was considered to be more than moderately effective in reducing the software development time (mean = 3.63 and 3.90). The results in the case of India were significant at the 0.01 α level.

One of the respondents claimed that dividing software into objects and then joining them together depends on client proprietary rights i.e., non-disclosure. Non-disclosure prevents full reuse of existing software from client to client rather than from project to project.

Thus, based on the factors mentioned in the questionnaire and the response obtained, it can be concluded that object oriented approaches are perceived to be effective in reducing the software development time.

III. Use of Reusable Code and Application Packages

Application packages and reusable code have only recently gained widespread use. It has been cited in the literature to be a relatively new and effective strategy in reducing the software development time.

As seen from Table 4, the use of reusable code and application packages in software development was perceived to be more than moderately effective in reducing the software development time. 60% and 50% projects in Canada and India respectively used this approach. 50% of the respondents in both countries said that they had a software code library in their firm. The mean value of 4.30 (significant at 0.01 α level) and 3.80 (significant at 0.1 α level) for the strategy of using reusable code from the firms own inventory in Canada and India respectively indicates that this strategy was perceived to be

significantly more than moderately effective in expediting software development. The mean of 3.92 and 3.38 for Canada and India respectively for the strategy of using application packages available in the market indicates that it was perceived to be moderately effective. The result was significant at 0.05 α level for Canada.

The respondents said that this strategy helped vastly in reducing the software development time because it eliminated the time that would have been spent developing software that already existed.

IV. Use of Incremental Development Approach

Use of incremental development approach was found to be an effective strategy in speeding up the software development process. According to the literature (Phan, Vogel and Nunamaker, 1995) this approach is fast becoming one of the most effective strategies in expediting software development.

Incremental development approach was used in 70% and 75% software projects in Canada and India respectively. This strategy was perceived to be significantly effective by the respondents (Mean = 3.92 and 4.60) at α levels 0.05 and 0.01 respectively.

It is hypothesized that the use of incremental development approach varies with the size of project. The bigger the project, the more iterations were applied while developing the software.

V. Use of CASE/Automated Tools

CASE/Automated tools are cited in the literature to be widely used in the software industry for expediting software development. According to the literature, there are various tools that are used during different stages of software development. However, this study did not find these tools as widely used as the literature states. This was indicated by the high percentage of not applicable responses. Only 25% and 40% projects in Canada and India respectively used this approach.

The use of CASE/Automated tools in the requirements, design and coding stage were all perceived to be effective in reducing the software development time by the Indian respondents. This was indicated by the mean, significantly different from 3 for all the three stages for Indian firms. However, the percentage of use was only 40%, so this result cannot be considered a real indication of its effectiveness. For the Canadian firms, the mean value of less than 3 indicates that the use of this approach was less than moderately effective. It was claimed that the use of these tools may sometimes prove to be more time consuming because of the tools related training that has to be given to the programmers. It

can be hypothesized that Canadian firms felt that it was better to use human expertise rather than to rely on automated tools which more often than not were out-dated.

VI. Use of Advanced Project Management Techniques

This section deals with the various project management techniques that are being used in the industry today in order to reduce the software development time. It also considers planning and use of scheduling techniques. Literature claims that no software can be successful unless it has good project management. These techniques can prove crucial to the success of the software and also to the schedule and cost budgets. One respondent summed it up by saying “Project Management is the key to *everything*”.

80% and 65% projects in Canada and India respectively used project management techniques. As seen from Table 4, many of the project management techniques were perceived to be more than moderately effective in reducing the software development time. Use of scheduling charts and schedule tracking tools (mean = 3.82, 4.00 and 3.82, 3.80 respectively for Canada and India respectively) were considered to be significantly (at α levels 0.05, 0.1, 0.05, and 0.1 respectively) more than moderately effective in reducing the software development time. Shifting employees from one area to another was perceived to be moderately effective in reducing the software development time (mean = 3.64 and 3.10). The mean of 3.64 for Canada turned out to be significant at α level 0.05.

Mixing of technical and non-skilled staff within the project was perceived to be moderately effective (mean = 3.09 and 3.00). This result was not significant.

Being innovative (mean = 4.43 and 4.40) was perceived to be significantly more than moderately effective at α level 0.01 in reducing the development time. Seeking new ideas from team members obtained a mean of 4.50 and 4.40 for Canada and India respectively. These results were significant at 0.01 α level. It can be concluded that being innovative and open to new ideas is perceived to be significantly very effective in reducing the software development time. Software being a fast growing industry requires the firms to be alert all the time and keep up with the changing technologies. It can be hypothesized that being innovative and open to new ideas will help firms keep up with the market.

Creating a sense of urgency in the project (mean = 4.00 and 3.10 for Canada and India respectively) helps expedite software development. The result was significant at α level 0.01 for Canada. Responding immediately to problems (mean = 3.93 and 4.70) instead of delaying it until after the software has been fully developed was perceived to be significantly more than moderately effective at α levels 0.05 and 0.01 for Canada and India respectively in expediting software development.

Motivating the team members by giving rewards was perceived to be moderately effective (mean = 2.92 and 4.70) in reducing the software development time. The Indian respondents found it to be significantly very effective at α level 0.01. Some of the

respondents said that members worked for their own satisfaction and this fact helped in motivating them. However, they claimed that a pat on the back for a job well done did make them feel good.

The use of different leaders in successive software increments was perceived to be less than moderately effective in reducing the software development time (mean = 2.20 and 2.50). A project leader has to build up a rapport with the members and switching them mid-project would lead to problems and confusion.

Checking for faults (mean = 4.29 and 4.40) at each stage of the software development process and correcting them immediately was perceived to be significantly more than effective at α level 0.01. Defining milestones early on in the project was also perceived to be significantly more than effective at α level 0.01 in reducing the software development time (mean = 4.43 and 4.30). Usually milestones were defined for the project. Learning from mistakes made in previous projects also ranked significantly high in reducing the software development time (mean = 4.21 and 3.90). The result was significant at α level 0.01. Respondents said that they listed all the mistakes made in projects and consciously avoided repeating them in future projects.

VII. Use of Structured Methods and Techniques

Literature claims that no software can be successfully developed without the help of a software development model. This model can be a theoretical model used worldwide or it can be a model developed by the firm itself for its own particular needs.

The use of structured models and techniques in software development was perceived to be moderately effective in expediting software development. It was used in 50% and 75% projects in Canada and India respectively. The use of models such as the waterfall model, and spiral model enticed a mean value of 3.09 and 4.10 for Canada and India respectively. The result for India was significant at α level 0.01. The respondents responded with a mean value of 2.80 and 4.44 for Canada and India respectively to the use of structured model developed by the firm for its own use. The result was significant at α level 0.01 for India. This result indicates that more firms in India felt that following a structured software development model significantly expedites software development. The reason for this could be that the parent companies want their subsidiaries in India to follow a model suitable for conditions in India, and the lack of autonomy in the subsidiaries. It can be hypothesized that the Indian respondents found the use of the structured models effective in reducing the development time because it gives a conceptual order to the development process and thus reduces rework.

VIII. Verification of Requirements

Verifying whether the software is being built according to the requirements specifications is very important in developing an effective software. If the software is not built according to the requirements it was supposed to satisfy, then the development of the software is a meaningless job.

The effectiveness of verifying the requirements at various stages of the software development process was perceived to be significantly more than moderately effective at the α level 0.01 in reducing the development time (mean = 4.21 and 4.50). Respondents said that verifying requirements was crucial for not only a strong financial performance but also for maintaining the image of the firm.

IX. Other Strategies

The respondents were asked to list strategies that they used to reduce the software development time but were not in the questionnaire. The following strategies were listed:

- 1) Constant interaction between development, quality assurance and product management. In other words, parallel as opposed to serial development.
- 2) Comparing with similar projects and making necessary improvements.
- 3) Brainstorming within the group.
- 4) Customer involvement throughout the project.

- 5) Choosing high quality staff and providing adequate training to the employees.
- 6) Involvement of all development staff in the establishment of software requirements.
- 7) Constant monitoring of evolution in software industry and potentially competitive technologies, in order to factor in new information into the software development specifications.
- 8) 80/20 rule wherein just 80% of the features of the software are implemented. The remaining 20% of the requirements are just ignored and not processed. Of course this will result in the software not being able to satisfy all the requirements specified. However, it does help expedite software development and the respondent claimed that ignoring 20% of the requirements did not make a significant difference in the working of the developed software.

To summarize, the descriptive statistics related to each strategy gave an overall picture of how effective the respondents perceived these strategies in terms of expediting software development.

7. CONCLUSIONS

In this section, the conclusions of this study are presented, followed by the limitations, and benefits of the study.

This study found that most firms in the Canadian and Indian high technology industry impose some sort of conceptual order on the software development process. These firms were also aware that in this competitive and rapidly changing business environment, expediting the software development process is the key to survival.

All but one respondent claimed that they used a structured software development process. Technical, business and quality assurance staff were reported to be involved upfront, in order to streamline the software development process and to ensure that the products did not go through several rounds of iterations in the future. These firms felt that it was worth putting in more efforts in the beginning while determining the requirements in order to avoid more time-consuming and costly iterations later. Another interesting finding was that most firms reported that quality assurance (QA) plays a significant role throughout the development process. The role of QA was described to check the validity of the design, perform qualifying tests, and to test the final product. QA offers analysis of the project's progress, consultancy and improvement suggestions, a neutral look at the project or service to give another viewpoint. QA also ensures support service to provide confidence that the firm is delivering the right products, which are being built in the right

manner, on time and within budget. One respondent summed up the importance of QA by saying “QA is a part and parcel of any software project”.

India is struggling hard to develop bug-free software of the highest quality. Hence, they are considering a move towards heavy documentation which they treat as an integral part of quality. As one respondent said, “it is very important to document what you do, and do whatever you document”.

The most significant difference found in the software development models followed in Canada and India (Figure 4 and 5) was the greater use of prototypes in the Indian companies. Prototyping helps reduce software development time in large projects. However, it can lead to delays in smaller projects.

Another difference was the absence of feedback from the various stages of the model to the requirements stage in the model followed by Indian firms. This could be because the requirements specifications were more rigid and clear-cut in these projects.

The use of structured models and techniques in software development was also perceived to be moderately effective in expediting software development. It was used in 50% and 75% projects in Canada and India respectively. This result is for projects undertaken during the past two years.

To reduce the software development time, the following strategies were considered effective by a majority of the Canadian and Indian respondents:

- Application of Prototyping Approach
- Application of Object-Oriented Approach
- Project Management Techniques
- Verification of Requirements

According to Lai (1996), because the software market continually evolves, software applications must adapt to newer platforms, environments, and changing demands. This means that rapid development processes for evolving software will continue to emerge on the basis of such factors as:

- *maintainability* - software must be quickly changeable or extensible to include new features,
- *productivity* - software products must be developed quickly to meet the changing market and to cut development cost,
- *reusability* - development time and effort for evolving products can be saved by reusing existing software artifacts,
- *reliability* - testing processes which will be affected by shorter production times, and
- *safety* - required for use in embedded, safety-critical systems and consumer products.

BENEFITS AND IMPLICATIONS FOR MANAGERS

This study on software development provides a good foundation for future detailed research in the area which has only been scantily explored. The information gathered in this study provides a detailed account of the software development process and the strategies used to reduce the software development time.

This study will help in obtaining an overview of the software development process models followed in Canada and India. Managers of new and less successful firms can use these software development models and improve their development process and restructure their activities and policies in order to gain competitive advantage. The models show the various stages that are followed in the development of software and also the linkages that exist among these stages. Managers can use these models as a basis for their development process and change the model to suit their particular requirements.

This study reveals the effectiveness of the various strategies used to expedite software development (Table 4). This list of strategies will provide a good guideline to both new and less successful firms in their efforts to expedite their software development process for improvement. Managers may use this table as an indicator to the effectiveness of the strategies and depending on the resources of their firm adopt the strategies. However, it should not be assumed that adopting only a few of the highest ranked strategies will

provide most impact on expediting software development. Care should be taken to adopt strategies that suit the firm's management philosophy and also the resources in terms of finance and labor.

LIMITATIONS

This study provides the basis for generalization. 23 subjects were considered and results analyzed. Ottawa region and Bangalore region based units were considered for study. However, the limitation on sample size restricted the scope for rigorous statistical analysis. Most of the data collected for this study was qualitative. There is a possibility that halo effect was introduced in responses to the instrument used in this study. Since this study relies mostly on qualitative data, there was a very high reliance of the researcher's judgment.

FUTURE RESEARCH

This study was an exploratory study and can be used as a basis for future research in the area. One of the areas that was not explored in this research was the effectiveness of the software models developed. This research developed the models being used in the high technology firms but, it was not studied whether these models were successful or not. The pointers to success that possibly can be used are product's economic success and the firm's ranking on Capability Maturity Model (CMM).

Research can also be conducted to study whether the development models followed in a single firm are different for different projects. Studies can also be conducted to find the differences in models with respect to the type of software; systems or applications.

Another area that can be explored is the cost-time tradeoff that exists in adopting the strategies to expedite software development.

LITERATURE CITED

Abbott R.J., *An Integrated Approach to Software Development*, John Wiley and Sons Inc., U.S.A., 1986.

Baber R.L., *The Spine of Software*, John Wiley and Sons Inc., U.S.A., 1987.

Bandinelli S., Fuggetta A., Lavazza L., Loi M., and Picco G.P., "Modeling and Improving an Industrial Software Process", *IEEE Transactions on Software Engineering*, Vol. 21, No.5, May 1995, pp.440-453.

Bennetan E.M., *Software Project Management*, Mc-Graw Hill Book Company, Berkshire, England, 1992.

Berkeley D., *Software Development Project Management*, Ellis Horwood Limited, Cornwall, Great Britain, 1990.

Bersoff E.H., and Davis A.M., "Impacts of Life-Cycle Models on Software", *Communications of the ACM*, Vol. 34, No.8, August 1991, pp.104-117.

Bhandari I., Halliday M., Tarver E., Brown D., Chaar J., and Chillarege R., "A Case Study of Software Process Improvement During Development", *IEEE Transactions on Software Engineering*, Vol. 19, No.12, December 1993, pp.1157-1170.

Bischofberger W. and Pomberger G., *Prototyping-Oriented Software Development*, Springer-Verlag Berlin Heidelberg, U.S.A., 1992.

Blum B. I., *Software Engineering A Holistic View*, Oxford University Press, 1992.

Blum B. I., "A Taxonomy of Software Development Methods", *Communications of the ACM*, Vol. 37, No. 11, November 1994, pp. 82-94.

Boar B., "Application Prototyping: A Life Cycle Perspective", *Journal of Systems Management*, February 1986, pp.25-31.

Boehm B.W., and Ross R., "Theory-W Software Project Management: Principles and Examples", *IEEE Transactions on Software Engineering*, Vol. 15, No.7, July 1989, pp.902-915.

Boehm B.W., "A Spiral Model of Software Development and Enhancement", *IEEE Engineering Management Review*, Winter 1995, pp.69-81.

Booch G., *Object Oriented Analysis and Design with Applications*, The Benjamin/Cummings Publishing Company, 1994.

Burch J.G., *Systems Analysis, Design and Implementation*, Boyd and Fraser Publishing Co., Boston, U.S.A., 1995.

Carmel E., "Cycled Time in Packaged Software Firms", *Journal of Product Innovation Management*, Vol. 12, No. 2, March 1995, pp. 110-123.

Dalal S., Horgan J. R., and Kettenring J. R., "Reliable Software and Communication II: Controlling the Software Development Process", *IEEE Journal on Selected Areas in Communications*, Vol. 12, No.1, January 1994, pp.33-39.

Davis A.M., and Sitaram P., "A Concurrent Model of Software Development", *Software Engineering Notes*, Vol. 19, No.2, April 1994, pp.38-51.

Deitz D., "Programming From a Clean Slate", *Mechanical Engineering*, April 1995, pp.84-86.

DeMarco T., and Lister T., *Software State-of-the-Art*, Dorset House Publishing, New York, U.S.A., 1990.

Easteal C. and Davies G., *Software Engineering*, Mc-Graw Hill Book Co. Ltd., Berkshire, England, 1989.

Eisenhardt K. M. and Tabrizi B. N., "Accelerating Adaptive Processes: Product Innovation in the Global Computer Industry", *Administrative Science Quarterly*, Vol. 40, March 1995, pp.84-110.

Garg P.K. and Jazayeri M., *Process-Centered Software Engineering Environments*, IEEE Computer Society Press, Los Alamitos, California, U.S.A., 1996.

Garlan D. and Perry D.E., "Introduction to the Special Issue on Software Architecture", *IEEE Transactions on Software Engineering*, Vol. 21, No.4, April 1995, pp.269-274.

Genuchten M. V., "Why is Software Late? An Empirical Study of Reasons for Delay in Software Development", *IEEE Transactions of Software Engineering*, Vol. 17, No.6, June 1991, pp.582-589.

Gupta A. K. and Wilemon D. L., "Accelerating the Development of Technology-Based New Products", *California Management Review*, Vol. 32, No.2, Winter 1990, pp.24-44.

Humphrey W. S., "Characterizing the Software Process: A Maturity Framework", *IEEE Software*, Vol. 5, No. 2, March 1988, pp. 73-79.

Hunt N., "Unit Testing", *Journal of Object-Oriented Programming*, February 1996, pp.18-23.

Jacobson I., and Christerson M., "A Confused World of OOA and OOD", *Journal of Object-Oriented Programming*, September 1995, pp.15-20.

Koenig A., "Compatibility vs. Progress", *Journal of Object-Oriented Programming*, February 1996, pp.48-50.

Lai R. C.T., "Extended Software Development Process", *Computer*, Oct. 1996.

Lewis T.G., *CASE: Computer-Aided Software Engineering*, Van Nostrand, New York, U.S.A., 1991.

Livari J., "Hierarchical Spiral Model for Information System and Software Development. Part 2: Design Process", *Information and Software Technology*, Vol. 32, No.7, September 1990, pp.450-458.

Maras M.G., Riel E. J., Nowlan D. R., Menzel L. J., and Mahoney M. J., "A Rapid Prototyping and Integrated Design System (RAPIDS) for Software Development of GN&C Systems", *Advances in the Astronautical Sciences*, Vol. 86, 1994, pp.89-102.

Mathur S., *Study of the Innovation Process in the Canadian Electronic and Telecommunication Industry: An Innovation Time Perspective*, Master of Management Studies Thesis, Carleton University, 1993.

Metzger P. W., *Managing a Programming Project*, 2nd Edition, Prentice-Hall Englewood Cliffs, New Jersey, 1981.

Mintzberg H., "An Emerging Strategy of 'Direct' Research", *Administrative Science Quarterly*, Vol.24, No.4, 1979, pp.582-589.

Mintzberg H., Raisinghani D., and Théorêt A., "The Structure of 'Unstructured' Decision Processes", *Administrative Science Quarterly*, Vol.21, No.2, 1976, pp.246-275.

Orlikowski W.J., "CASE Tools as Organizational Change: Investigating Incremental and Radical Changes in Systems Development", *MIS Quarterly*, September 1993, pp.309-340.

Pang C. Y., "A Software Development Approach Using a Base Architecture, Cluster Patterns, and Surrogates", *Journal of Object-Oriented Programming*, February 1996, pp.24-32.

Parker C, and Case T., *Management Information Systems*, McGraw-Hill, Inc. U.S.A., 1993.

Patton M. Q., *Creative Evaluations*, Sage Publications Inc., California, 1981.

Phan D.D., Vogel D. R., and Nunamaker J. F., "Empirical Studies in Software Development Projects: Field Survey and OS/400 Study", *Information and Management*, Vol. 28, No 4, April 1995, pp.271-280.

Pilkington R., "The Life-Cycle Approach to Software Development", *Manufacturing Chemist*.

January 1996, pp.28-33.

Pressman R.S., *A Manager's Guide To Software Engineering*, Mc-Graw Hill Inc., U.S.A., 1993.

Rauscher T. G., and Smith P. G., "From Experience - Time-Driven Development of Software in Manufactured Goods", *Journal of Product Innovation Management*, Vol. 12, No. 3, June 1995, pp. 186-199.

Rosenau M. D. Jr., and Lewin M.D., *Software Project Management*, Lifetime Learning Publications, Belmont, CA, 1984.

Rumbaugh J., "Models for Design: Generating Code for Associations", *Journal of Object-Oriented Programming*, February 1996, pp.13-17.

Samyukta Karnataka, News Article, Aug. 1996.

Shina S. G., *Concurrent Engineering and Design for Manufacture of Electronics Products*, Van Nostrand Reinhold, New York, 1991.

Special feature, "Where is Software Headed? A Virtual Roundtable", *Computer*, August 1995, pp. 20-32.

Srivastava U. K., Shenoy G. V., and Sharma S. C., *Quantitative Techniques for Managerial Decision Making*, Wiley Easter Limited, New Delhi, 1983.

Stevens W., *Software Design*, Prentice Hall International, U.K., 1991.

Urban G. L., and Hauser J. R., *Design and Marketing of New Products*, Prentice-Hall Inc., Englewood Cliffs, New Jersey, 1980.

Vonk R., *Prototyping*, Prentice Hall, Hertfordshire, U.K., 1990.

APPENDIX

APPENDIX I

REQUEST LETTER GIVEN TO RESPONDENTS PRIOR TO THE INTERVIEW

Carleton University
School of Business
Ottawa, ON K1S5B6

Dear Sir/Madam

Re: Research on Software Development Process

Our graduate student, Suparna Kurdi is conducting a research on the software development process in the high technology industry in fulfilment of her master's thesis requirement under my supervision. This study has two objectives: the first is to understand the software development process; the second objective is to generate a comprehensive list of strategies considered important to reduce the software development time. As your firm is one of the leading high technology firms, your participation in this study would be greatly appreciated.

We are seeking an interview which will last no more than one hour. This session will include open ended questions where you will be asked to describe the software development process for one of the recent projects (last 2-3 years) for which you were responsible. Further, in order to have a comprehensive list of strategies considered important for expediting the software development process, you will be asked to complete a questionnaire after the interview session. The questionnaire takes approximately fifteen minutes to complete.

All interview responses will be kept confidential. If desired, a summary report will be made available to you.

I appreciate your assistance as your help is essential to conduct this study.

Yours Sincerely,

Dr. Uma Kumar

APPENDIX II

QUESTIONNAIRE FOR THE TWO RESEARCH OBJECTIVES

A: Questionnaire containing open ended questions for the first research objective, along with the probes used by the researcher. This questionnaire was retained by the researcher.

B: Self-administered questionnaire for the second research objective that was given out to the respondents to mark their responses.

QUESTIONNAIRE A:

Questions:

Conventions used in this interview schedule are:

1 Questions to be asked by the interviewer are in Lower Case

2 Probes are in Bold

3 Interviewer notes are in Upper Case

Q-1 The Firm -----

Q-2 Respondent's Name -----

Q-3 Respondent's Designation -----

Q-4 Division of firm -----

Q-5 Most of the firms follow a series of steps during software development

Could you please describe the software development process (starting from requirements specifications to implementation) followed for a software project undertaken in the last 2-3 years for which you were responsible ? (**what kind of decisions, major events were involved**). Further, this software should either be entirely new or should be a significantly modified/improved product over a previous/existing software. The software projects considered may either be external and internal projects. In the internal projects, the software will be developed for use within the firm. In the external projects, the software will be developed for a user not within the same firm. The kinds of software that are considered maybe structured and non-structured, application or systems or other software.

PROBES FOR THIS OPEN-ENDED QUESTION

- (a) Was the software you are considering an internal or external project?**
- b) Was the software requirements structured (TPS, Complex Application Software) or non-structured (DSS or ES)?**
- c) Was the software developed an application software or systems software or others?**
- d) Was the starting stage of the software development process the customer's (user or anybody who requests the development of the software) request for proposal along with their functional specifications? If not, what was the starting point of the development?**
- e) Did you have any stage where the system requirements, software requirements and feasibility factors were determined?**
- f) Is there a stage wherein Preliminary Design of the specific features of the software was performed?**
- g) Is there a stage wherein Detailed Design of the software to be developed was performed?**
- h) Was Coding and Debugging done during the development process?**
- i) Was Testing done after each stage or was it done once at the end after the software was developed?**
- j) Was Maintenance of the software done and was there a feedback to systems requirements?**

k) Does Quality Assurance play an important role throughout the development process?

i) Correspondence: Check for how well the delivered system corresponds to the needs of the operational environment.

ii) Correctness: Check the consistency of the product with respect to its specifications.

iii) Verification: Check to find out if the software is being built in the right manner, following proper guidelines.

iv) Validation: Check to find out if the software is not only working well but also that it is doing what it was built to do.

l) Do any software development standards exist?

m) Is the design reviewed to check if it follows the software development standards?

n) Is code inspected to check if it follows the software development standards?)

IT WAS ALSO DETERMINED WHETHER THERE WAS ANY OTHER STAGE MENTIONED BY THE RESPONDENT DURING THE COURSE OF THE INTERVIEW WHICH WAS NOT INCLUDED IN THE LIST SO THAT MORE DETAILS COULD BE ASKED. THE APRIORI MODEL WITHOUT LINKAGES WAS USED AND DEPENDING ON THE RESPONSES THE LINKS WERE ADDED TO THE MODEL. IF THE RESPONSES INDICATED AN ADDITION/DELETION IN THE STAGES OF THE PRESENT APRIORI MODEL, THE MODIFICATION WERE MADE.

FURTHER, TO INVESTIGATE THE LINKS BETWEEN THE VARIOUS STAGES OF THE MODEL, THE FOLLOWING QUESTIONS WILL BE ASKED.

PROBES TO BE USED

Was there a link between

LA) System requirements, and software requirements

LB) Software requirements, and preliminary design

LC) Preliminary design, and detailed design

LD) Detailed design, and code and debug

LE) Code and debug, and test and pre-operations

LF) Test and preoperations, and operations and maintenance

Q-6 Did any of the stages in your process have feedback loops?

IN ORDER TO DETERMINE THE FEEDBACKS THAT WERE PRESENT IN THE SOFTWARE DEVELOPMENT PROCESS FOLLOWED, THE FOLLOWING QUESTIONS WERE ASKED:

PROBES TO BE USED

Was there feedback between

LG) Software Requirements and System Requirements

LH) Preliminary Design and Software Requirements

LI) Detailed Design and Preliminary Design

LJ) Coding and Debugging and Detailed Design

LK) Testing and coding & Debugging

LL) Maintenance and Testing

LM) Preliminary design and system requirements

LN) Detailed Design and System Requirements

LO) Code and Debug and System Requirements

LP) Testing and System Requirements

LQ) Maintenance and System Requirements

Besides the above mentioned stages, were there any other feedback loops in the process?)

Are Prototypes used during the Software development process. If used are they used at the following stages?

LR) Systems Requirements

LS) Software Requirements

LT) Preliminary Design

LU) Detailed Design

LV) Code and Debug

Were prototypes used at any stage other than those above?

Thank you very much for your time for the interview, Now I would like to ask you to kindly fill out a questionnaire.

QUESTIONNAIRE B:

Listed below are number of strategies that may affect the software development time.

Please rate the effectiveness of each of these strategies for reducing the software development time as experienced by you during the last two years of software development. **You are requested not to link these strategies with criteria for success of the software.**

1. Application of Prototyping Approach

Do you use prototypes? ☐ YES ☐ NO

If Yes, continue. If no, please go to Question 2.

What type of Prototype do you use ☐ Evolutionary ☐ Throwaway

	Not Effective (No change)		Moderately Effective		Very Effective	Not Applicable
Use of prototypes in system requirements stage	1	2	3	4	5	N/A
Use of prototypes in design stage	1	2	3	4	5	N/A
Use of prototypes in coding stage	1	2	3	4	5	N/A

2. Application of Object-oriented Approach

Segment the software requirements into various Sections and then integrating them	1	2	3	4	5	N/A
Divide software development into objects and then join them together	1	2	3	4	5	N/A

3. Use of Reusable Code and Application Packages

Do you have a software code/application library in your firm ☐ YES ☐ NO

Use of existing software code from own inventory	1	2	3	4	5	N/A
Use of software applications already available in the market	1	2	3	4	5	N/A
4. Use of Incremental Development Approach						
Compile, test, integrate, and run acceptance test at each stage of software development	1	2	3	4	5	N/A
5. Use of CASE/Automated Tools¹						
Use of CASE/automated tools in requirements stage	1	2	3	4	5	N/A
Use of CASE/automated tools in design stage	1	2	3	4	5	N/A
Use of CASE/automated tools in coding stage	1	2	3	4	5	N/A
6. Use of Advanced Project Management Techniques						
Use of Scheduling charts such as Gantt chart and PERT chart	1	2	3	4	5	N/A
Use of schedule tracking tools	1	2	3	4	5	N/A
Shift employees from one area to another within same project in order to get the best results as for time or cost	1	2	3	4	5	N/A
Mix technical and non-skilled work force in order to get the best results as for time or cost	1	2	3	4	5	N/A
Be innovative and put forth your ideas	1	2	3	4	5	N/A
Seek new ideas from team members	1	2	3	4	5	N/A
Create a sense of urgency for project among employees	1	2	3	4	5	N/A
Respond immediately to problems instead of doing it for once after the complete software is developed	1	2	3	4	5	N/A
Motivate team involved in the project with rewards if						

¹ CASE/Automated tools such as PacBase, SignOn, Turbo Analyst etc.

planned schedule is kept up	1	2	3	4	5	N/A
Use of different leaders for successive software increments	1	2	3	4	5	N/A
Check for faults at each stage and correct them	1	2	3	4	5	N/A
Define milestones early in the project	1	2	3	4	5	N/A
Improvements based on the learning from previous mistakes	1	2	3	4	5	N/A
7. Use of Structured Methods and Techniques						
Use of a software development model such as the Classic, Waterfall or Spiral model	1	2	3	4	5	N/A
Structured formal method of software development (particular to your firm)	1	2	3	4	5	N/A
8. Verification of Requirements						
Check throughout the development process whether the requirements are being satisfied	1	2	3	4	5	N/A
9. Other Strategies used						
Is there any other strategy for reducing time of development you would like to mention. If so, please use this space to do it.						
i	1	2	3	4	5	N/A
ii	1	2	3	4	5	N/A
iii	1	2	3	4	5	N/A

Please answer the following questions to enable us to find out the extent of use of the different strategies for reducing the software development time. Please use projects undertaken in the last two years as reference.

i) Percentage of projects in which prototypes were used _____%

ii) Percentage of projects in which object oriented approach was used _____%

iii) Percentage of projects in which existing code (from own inventory or market) was used _____%

iv) Percentage of projects wherein the software was developed in increments _____%

v) Percentage of projects in which CASE/automated tools were used _____%

vi) Percentage of projects wherein project management techniques were used _____%

vii) Percentage of projects in which a structured method of software development was adopted _____%

Your contribution to this effort is greatly appreciated. Would you like a summary of the results?

a. YES

b. NO

THANK YOU